Program Reasoning

13. Representation-based Search

Kihong Heo

KAIST

Dimensions in Program Synthesis

Specification (semantic constraint)

Input/output examples

...

Search space (syntactic constraint)

Domain-specific language

...

Search strategy

Enumerative search

Representation-based search

...
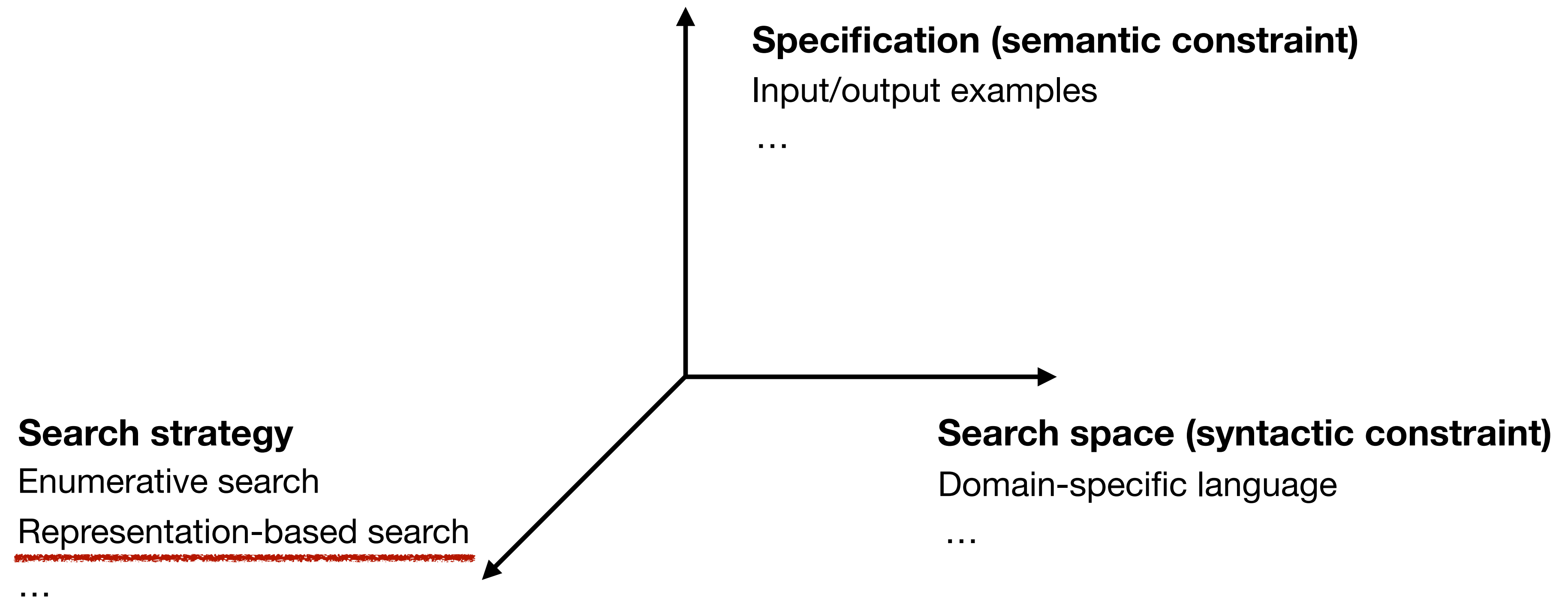
Goal: Finding a Set of Programs

- So far: search for a single solution

  - Enumerate one-by-one

- This lecture: search for a set of solutions

  - Return multiple results then rank them

  - Space-efficient search
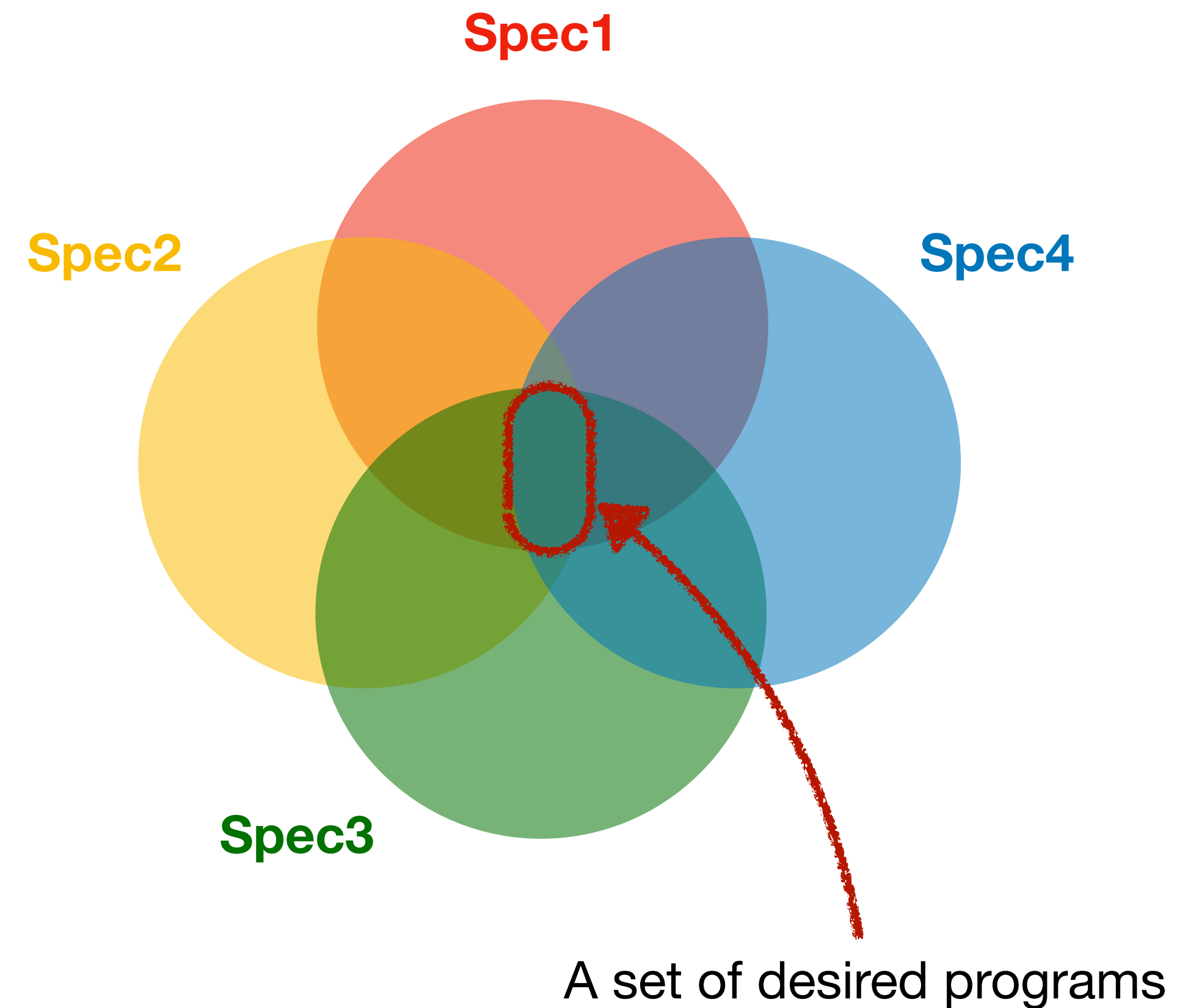
Spec1

Spec2

Spec4

Spec3

A set of desired programs

Representation-based Search

- Idea:

    - Build a data structure that concisely represents a set of programs

    - Extract solutions from that data structure

- Two well-known methods

    - Version space algebra (VSA)

    - Finite tree automata (FTA)

Version Space

- Hypothesis: a function that takes an input and an output

- Hypothesis space H: a set of all hypotheses (i.e. programs)

- Version space $VS_{H,D} \subseteq H$: a set of programs that satisfy the examples in the given dataset

  - $D = \{(in_i, out_i)\}_i$: a set of input-output examples

  - $h \in VS_{H,D} \iff \forall i, o \in D . h(i) = o$

Version Space Algebra

- A set of operations to manipulate and compose version space

- Operations on version spaces:

  - learn(i, o): construct a version space of functions consistent with (i, o)

  - $VS_1 \cap VS_2, VS_1 \cup VS_2$: intersection and union of two version spaces

  - pick VS: pick a function from version space VS

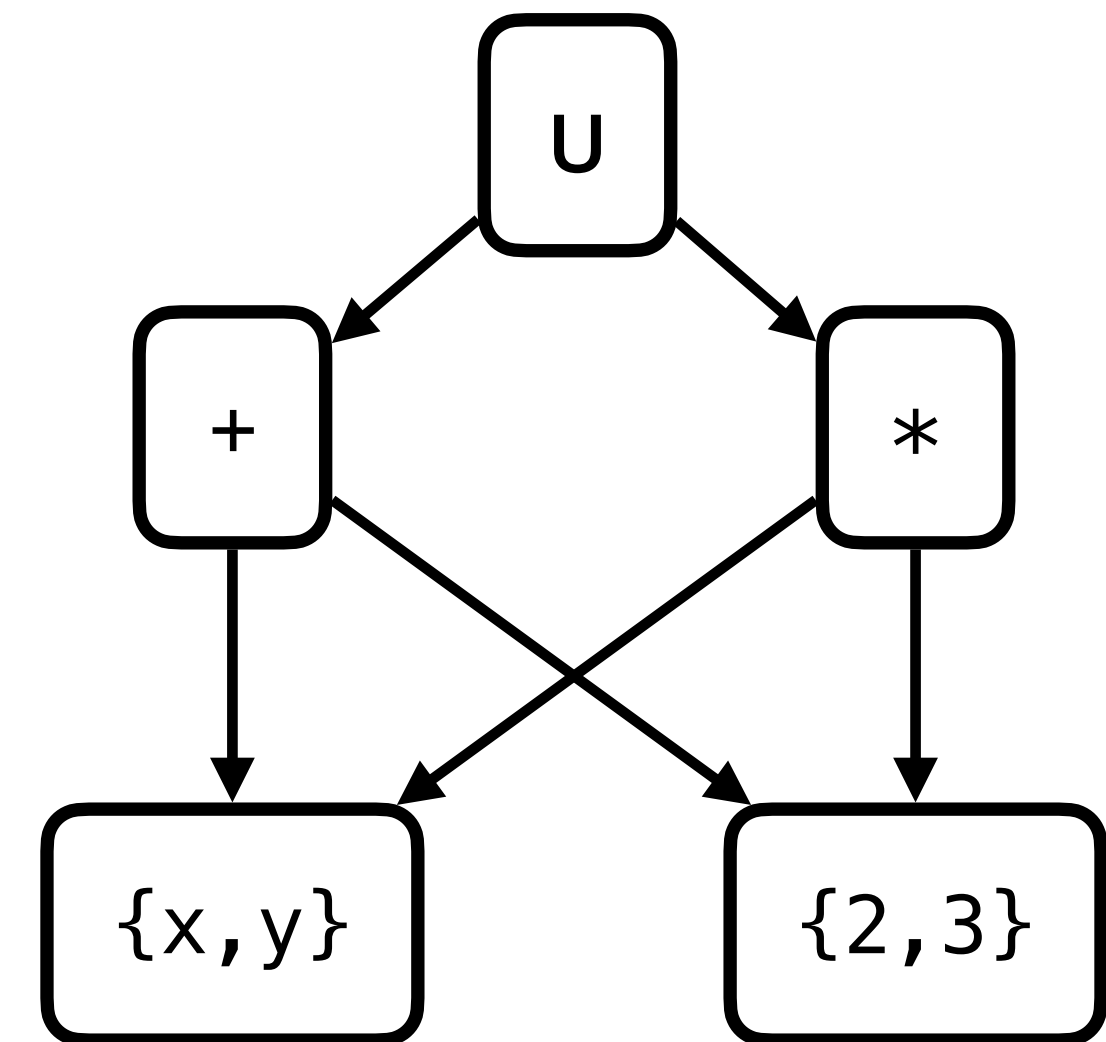- Synthesis idea: use of compact symbolic representation for the version spaces

Syntax of VSA

- Grammar of VSA

$$\widetilde{P} ::= \{P_1, \ldots, P_k\} \mid \mathbf{U}(\widetilde{P}_1, \ldots, \widetilde{P}_k) \mid F_{\bowtie}(\widetilde{P}_1, \ldots, \widetilde{P}_k)$$

- Example: {x+2, x+3, y+2, y+3, x∗2, x∗3, y∗2, y∗3}

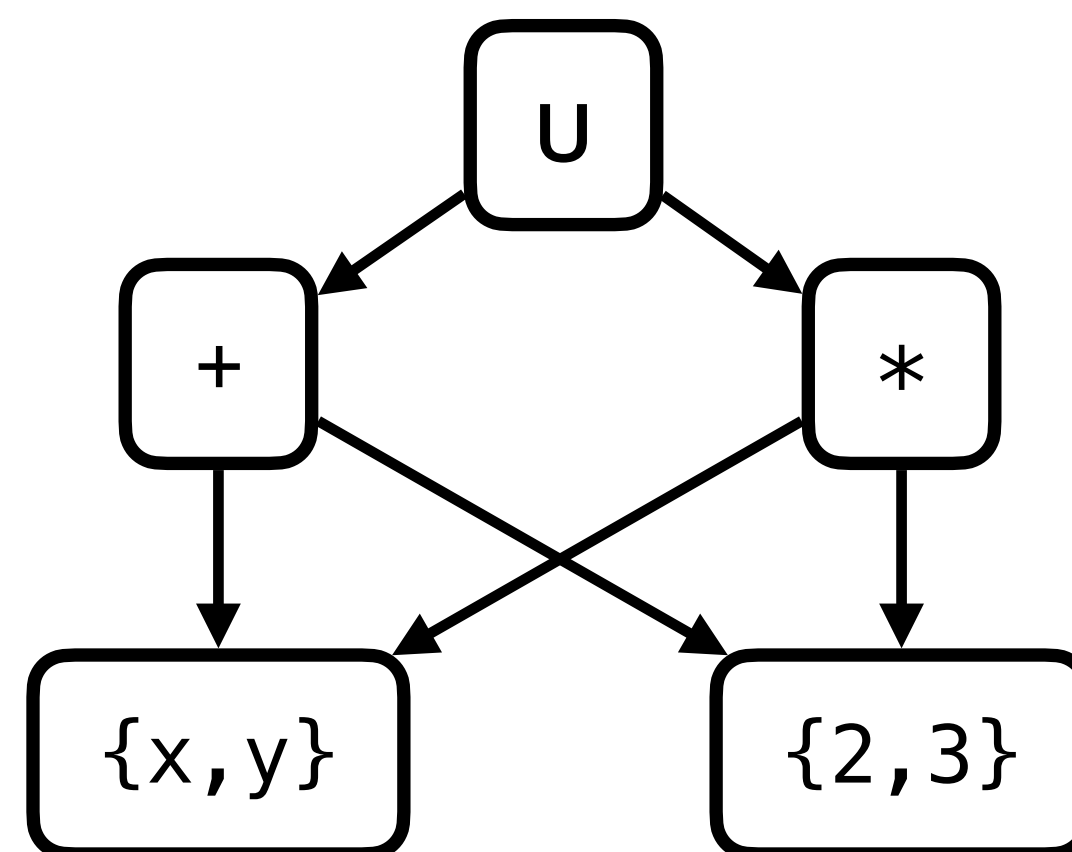$$\mathbf{U}(+_{\bowtie}(\{x,y\}, \{2,3\}), *_{\bowtie}(\{x,y\}, \{2,3\}))$$

Semantics of VSA

- A program P is an element of a VSA

$$P \in \{P_1, \ldots, P_k\} \qquad \exists j.P = P_j$$
$$P \in \mathbf{U}(\widetilde{P}_1, \ldots, \widetilde{P}_k) \qquad \exists j.P \in \widetilde{P}_j$$
$$P \in F_{\bowtie}(\widetilde{P}_1, \ldots, \widetilde{P}_k) \quad P = F(P_1, \ldots, P_k) \wedge \forall j.P_j \in \widetilde{P}_j$$
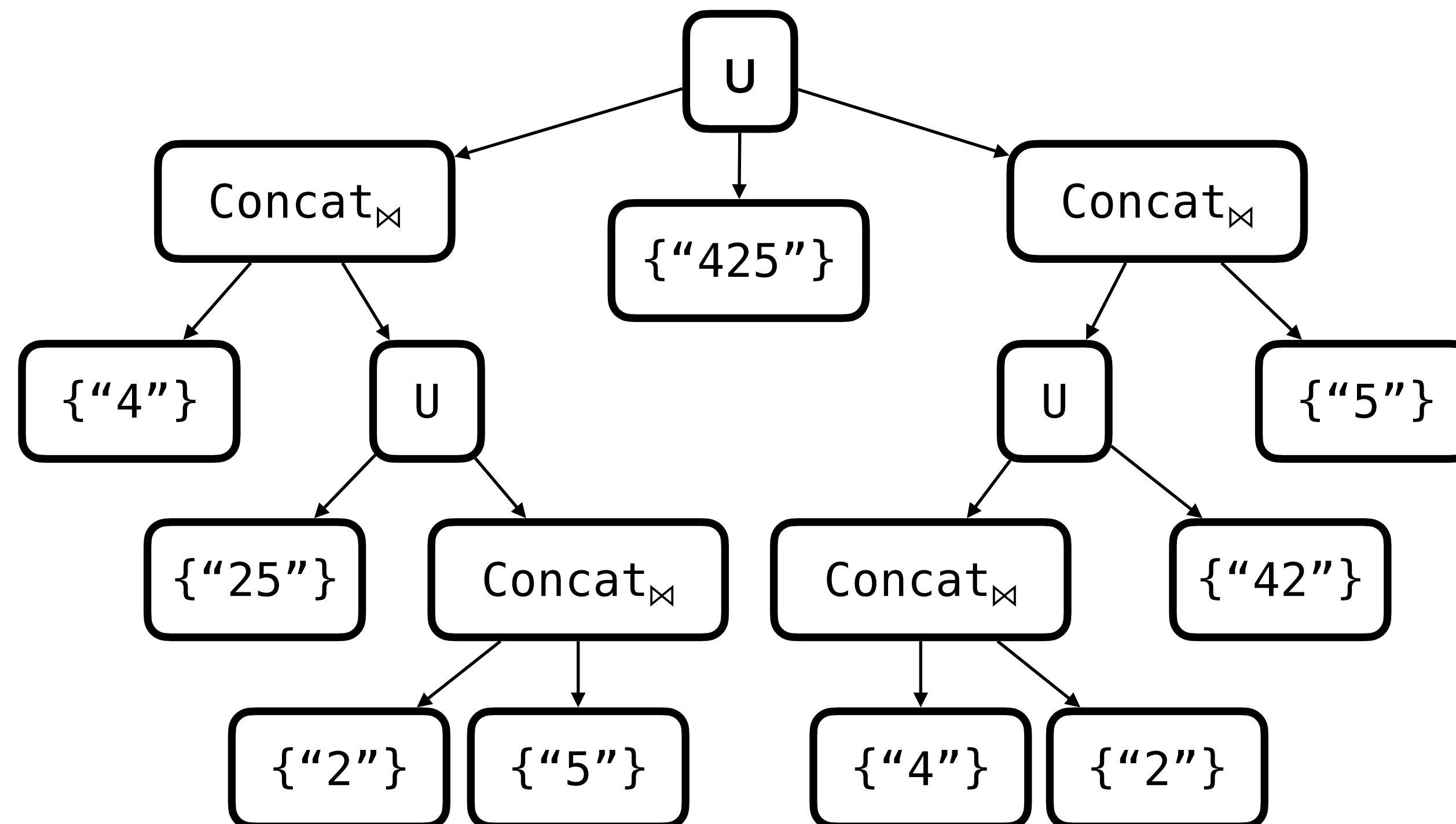
- Example:

Example
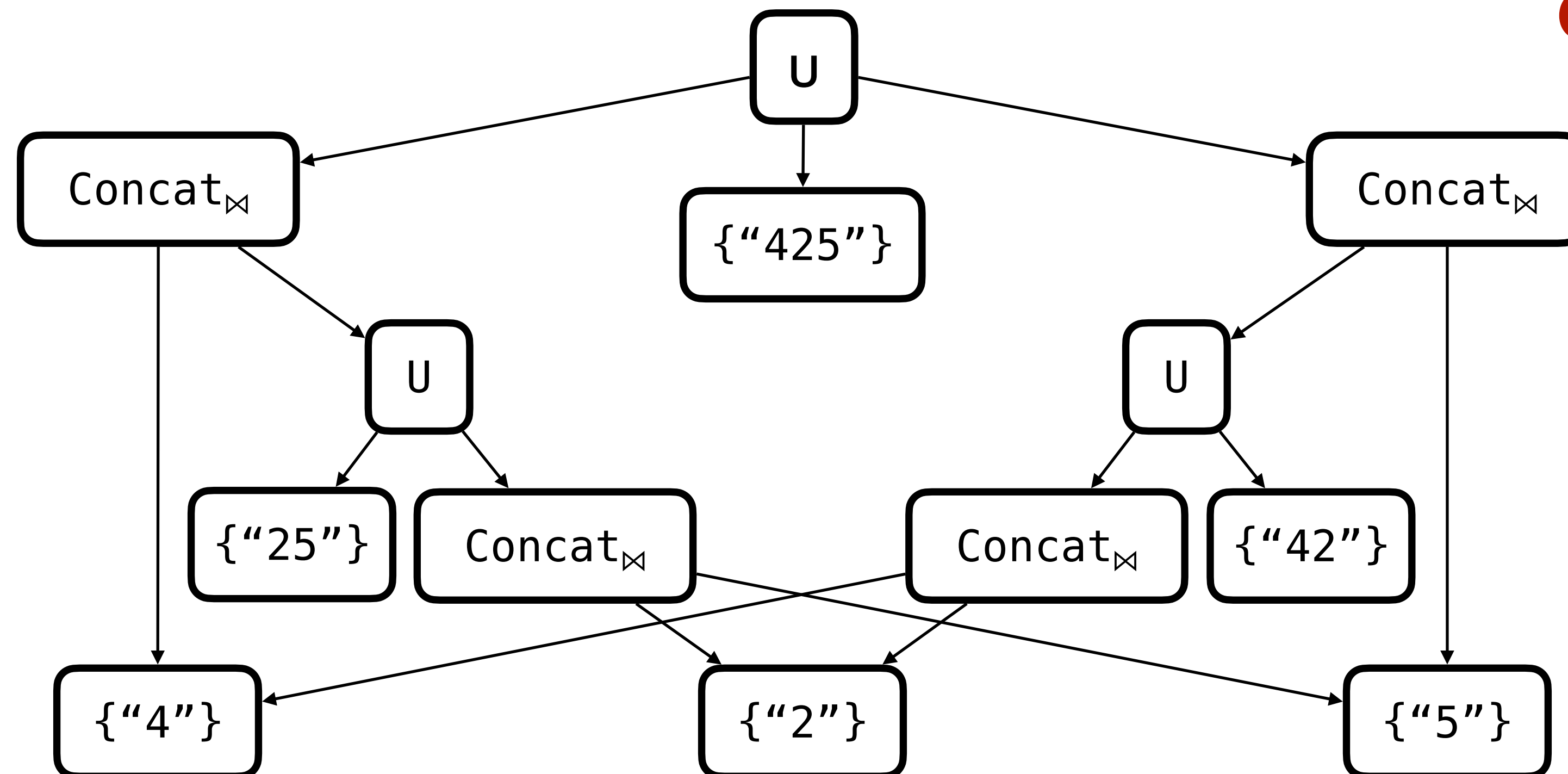
- Grammar $S \rightarrow ConstStr \mid \text{Concat}(S, S)$

- A set of program that returns "425"

Example

- Grammar $S \rightarrow ConstStr \mid \text{Concat}(S, S)$

- A set of program that returns "425"

Optimization!

Efficiency

- Represent potentially exponential program sets in polynomial space

  - V(VSA) : # nodes in VSA

  - |VSA| : # programs in VSA

  - V(VSA) = O(log|VSA|)

- E.g., millions of programs => hundreds of nodes

TDP with VSA

- Given a spec and a production, infer specs for subprograms (divide-and-conquer)

  - When $f<E_1, E_2, ..., E_n>$ (In)= Out where E_i is a subprogram

  - What is the spec for each E_i?
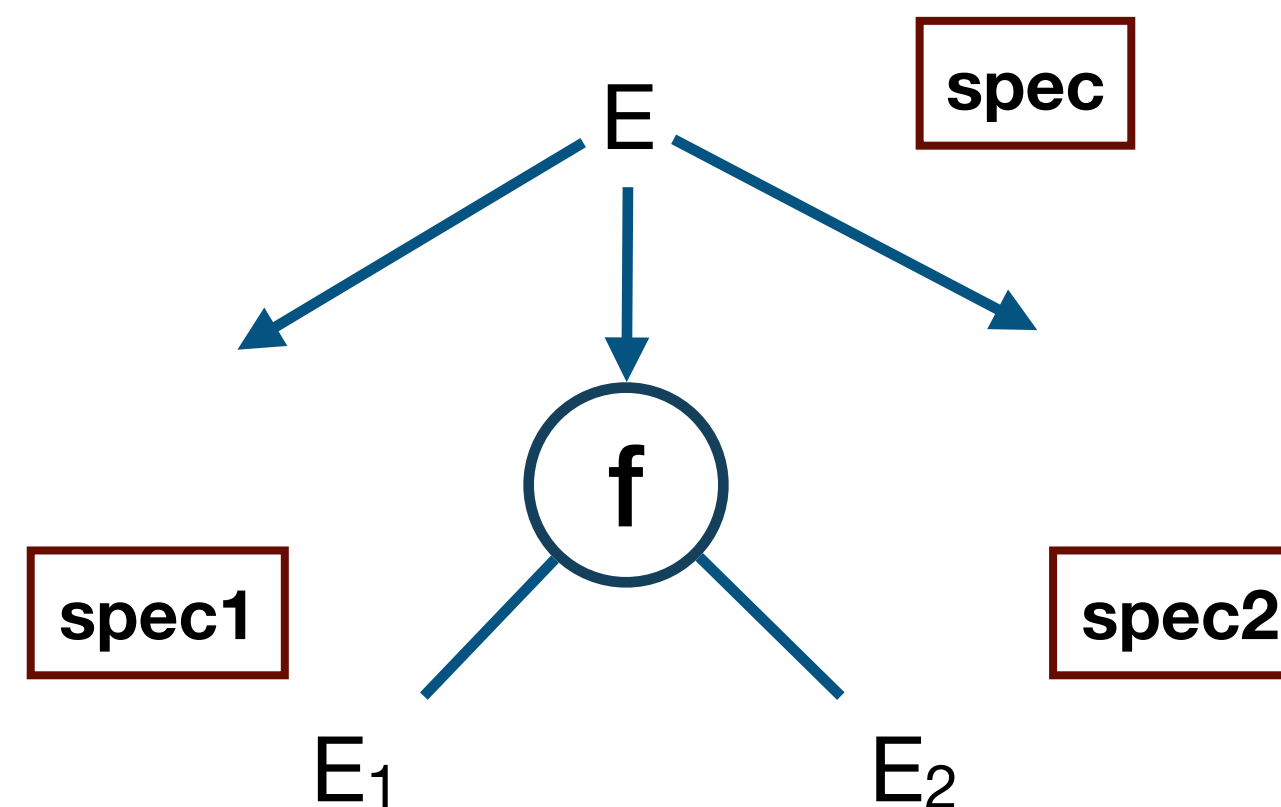
Example

- Grammar: $S \rightarrow ConstStr \mid x \mid \text{Concat}(S, S)$

- Specification: $f(\text{"SA"}) = \text{"USA"} \wedge f(\text{"AE"}) = \text{"UAE"}$

- Inverse set:

  - $\text{Concat}^{-1}(\text{"USA"}) = \{(\text{"U"}, \text{"SA"}), (\text{"US"}, \text{"A"})\}$

  - $\text{Concat}^{-1}(\text{"UAE"}) = \{(\text{"U"}, \text{"AE"}), (\text{"UA"}, \text{"E"})\}$
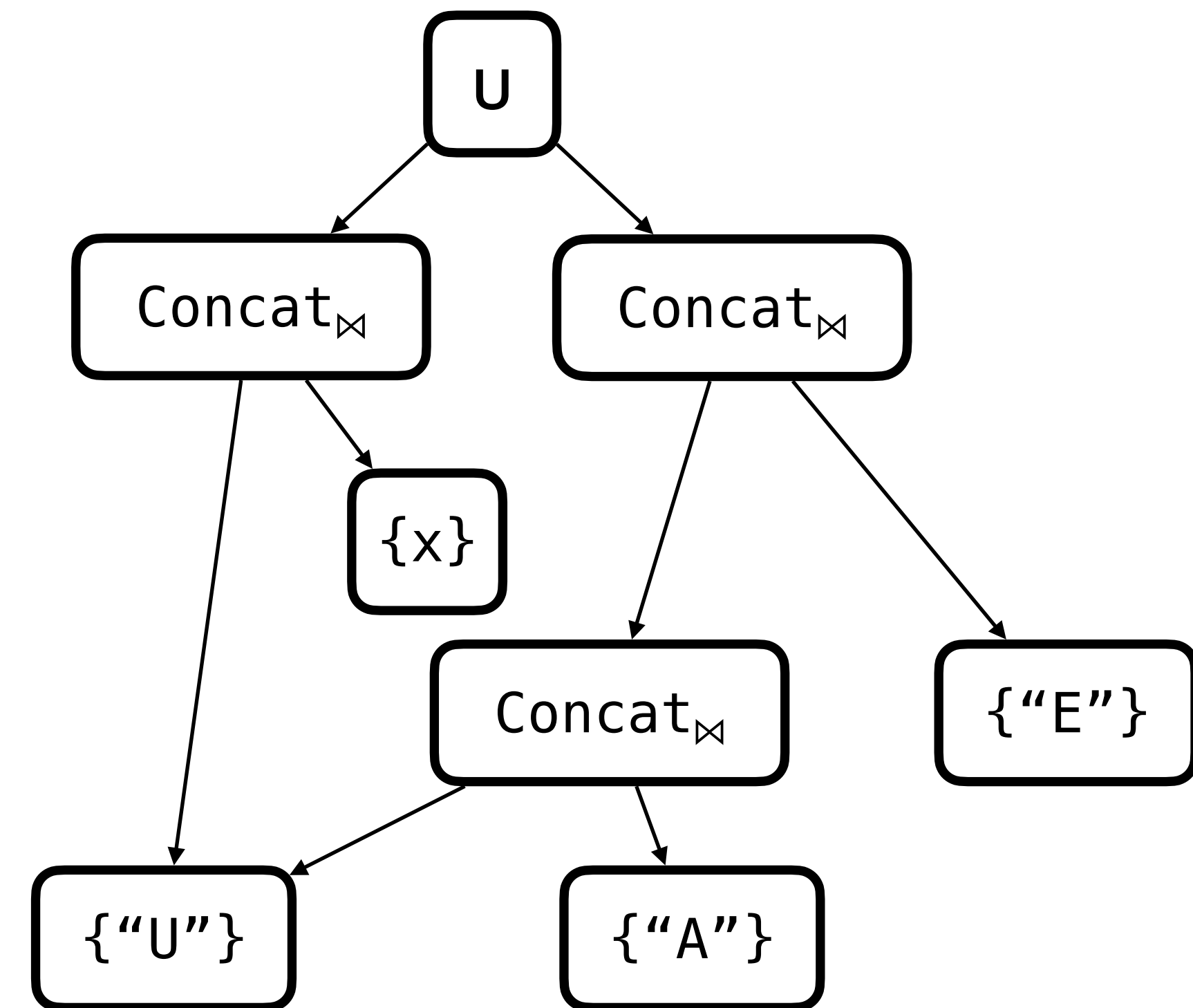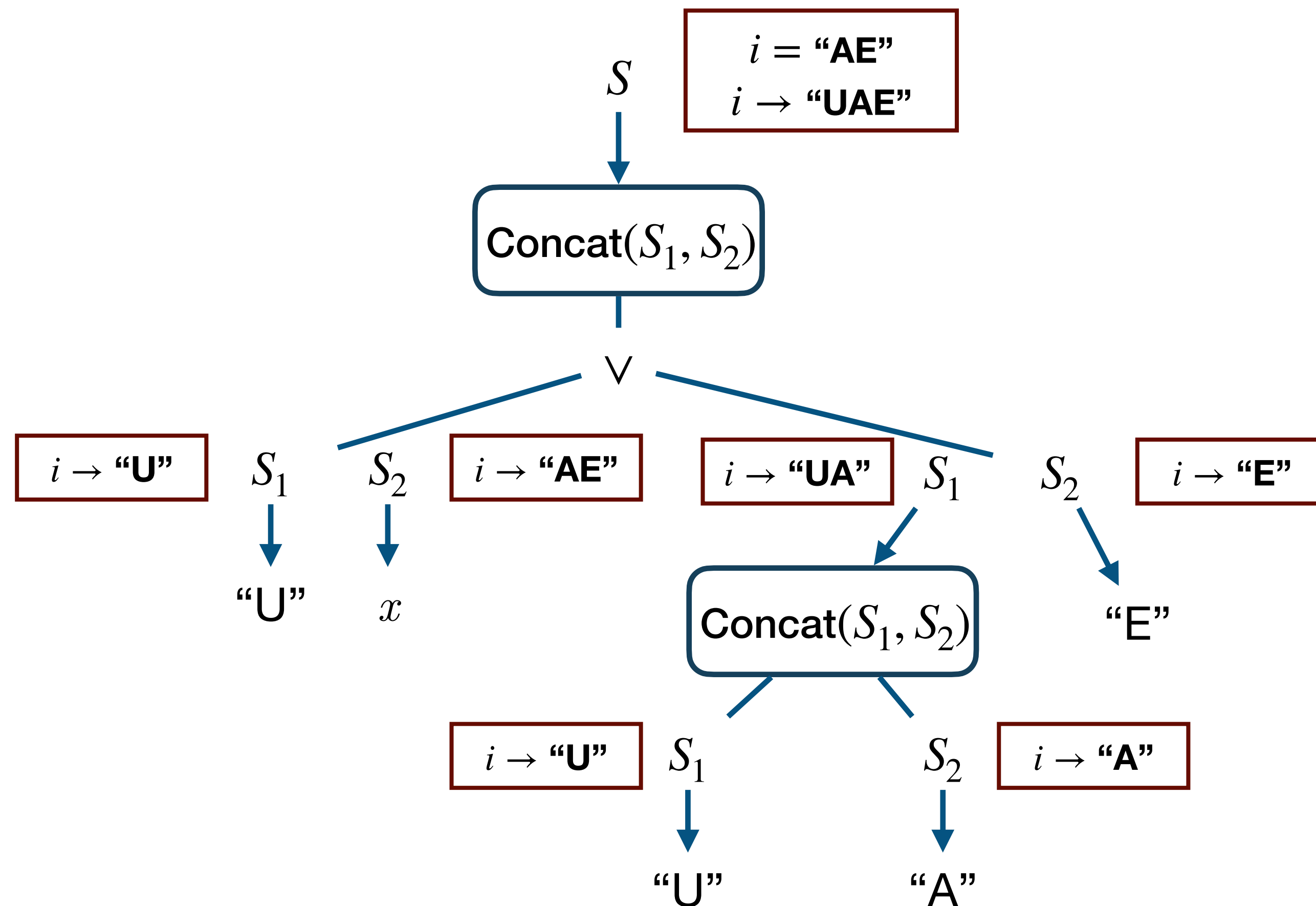
Step 1-1: Learn

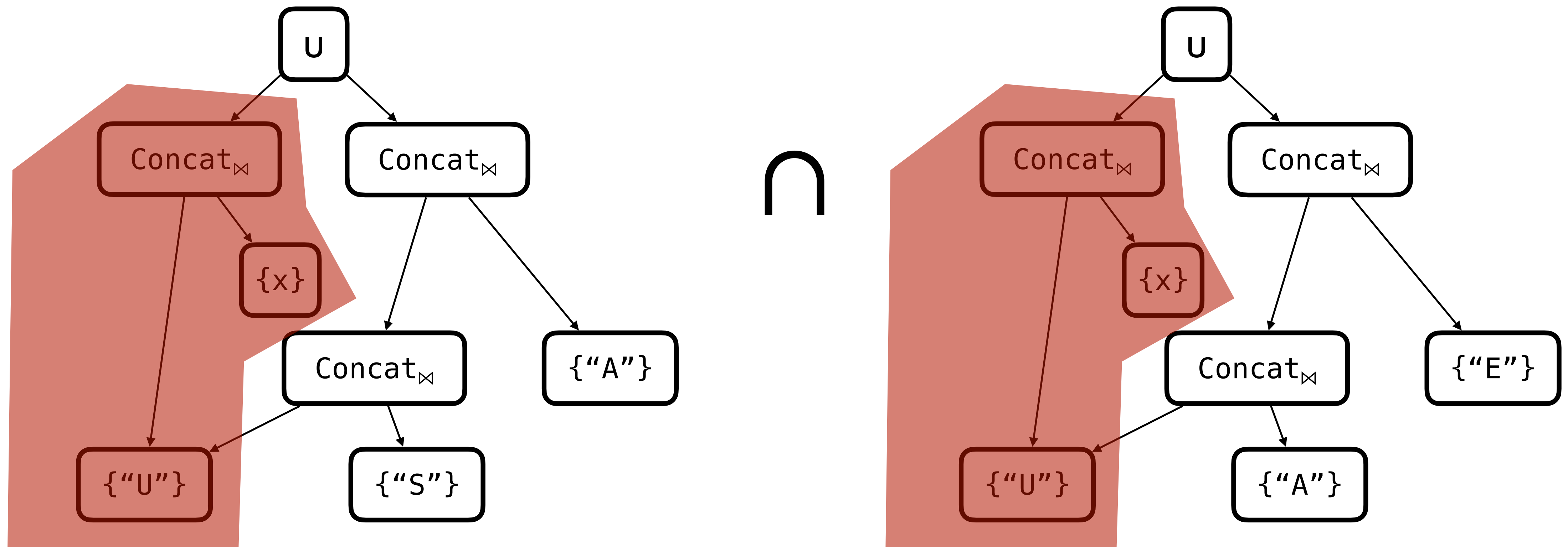- Top-down propagation with one example
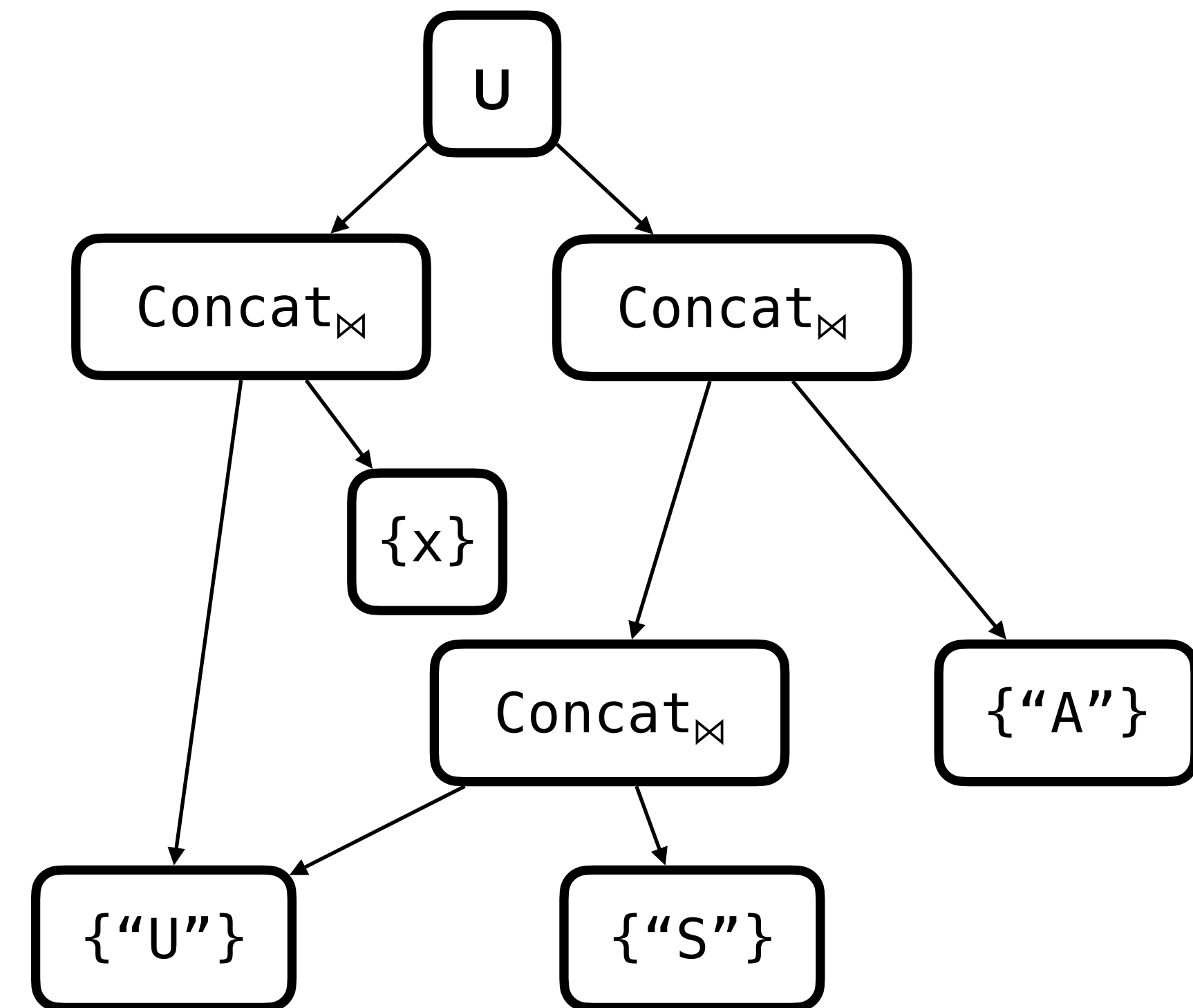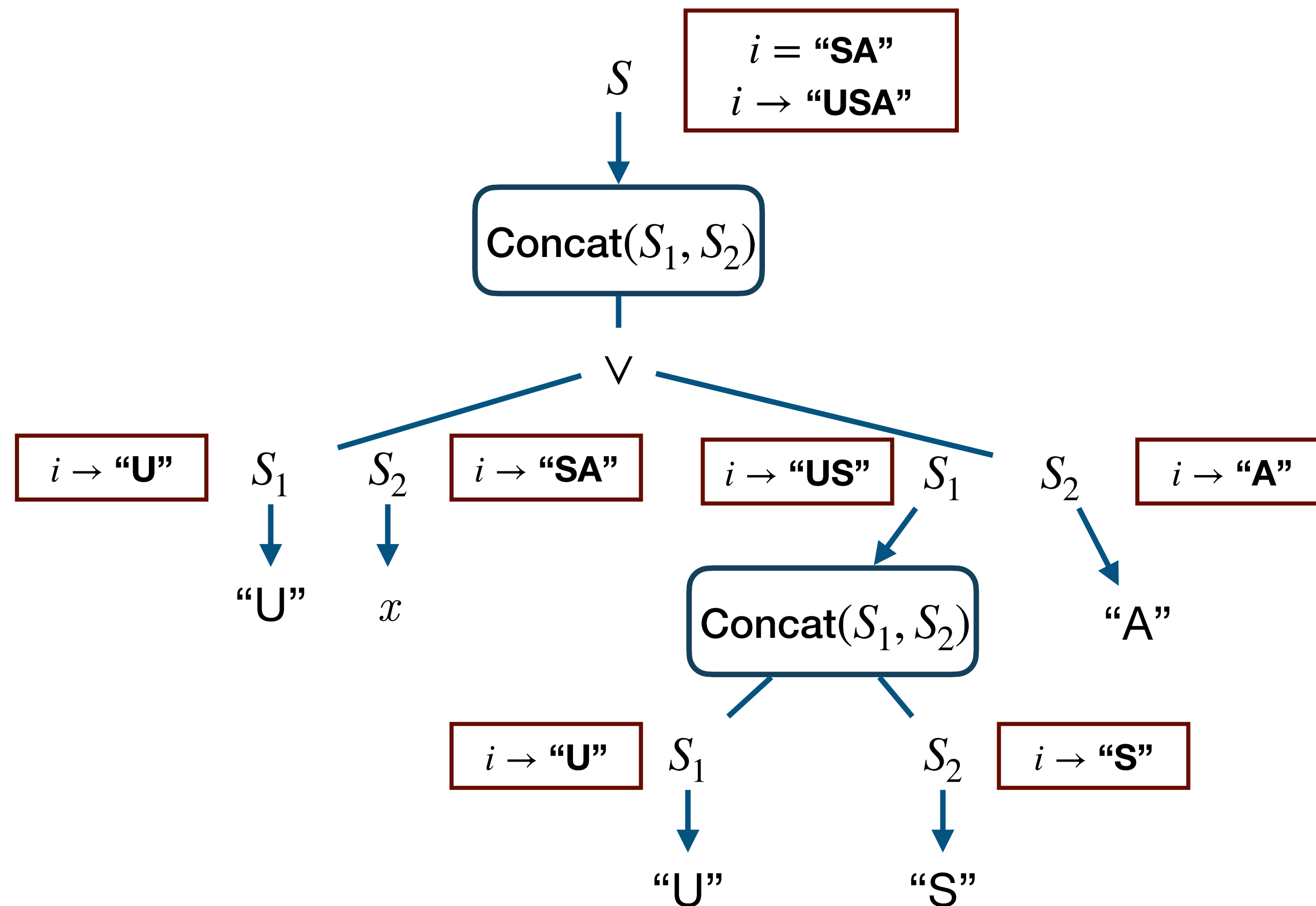
Step 1-2: Learn

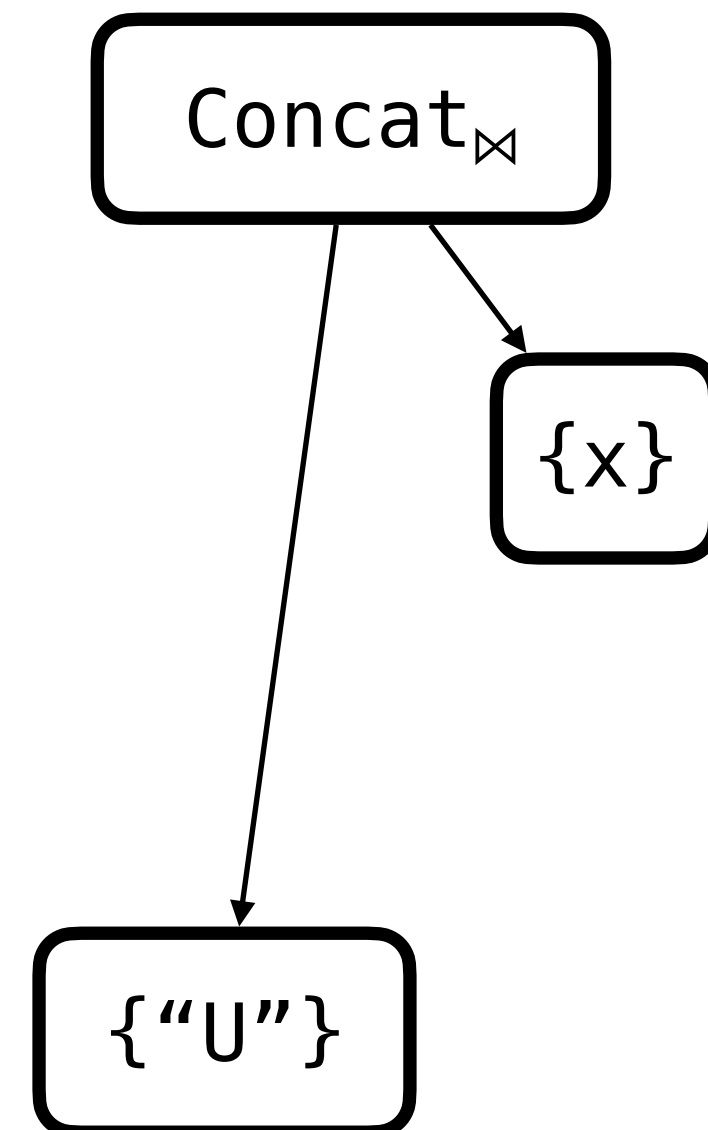- Top-down propagation with next example

Step 2: Intersection

- Intersection of two version spaces

Step 3: Pick

- Pick a desired program

```
Concat⋈
```

{x}

{"U"}

Concat(**"U"**, x)

Example

- Grammar:
  $S \rightarrow C \mid X \mid \text{Concat}(S, S) \mid \text{SubStr}(X, I, I) \mid \text{At}(X, I)$
  $I \rightarrow K \mid \text{IndexOf}(X, C, K) \mid \text{Length}(X)$
  $C \rightarrow \text{""} \mid \text{" "}$
  $X \rightarrow x$
  $K \rightarrow 0 \mid 1$

  > SubStr(s, i, n): longest substring of s of length at most n at i
  > E.g., SubStr("KAIST", 3, 5) = "ST"

- Specification: $f(\text{"Kihong Heo"}) = \text{"K Heo"} \wedge f(\text{"Gildong Hong"}) = \text{"G Hong"}$

- Solution: $f(x) = \text{Concat}((\text{At}(x, 0), \text{Substr}(x, \text{IndexOf}(x, \text{" "}, 0), \text{Length}(x)))$

- Inverse set: $\text{Concat}^{-1}(\text{"K Heo"}) = \{(\text{"K"}, \text{" Heo"}), (\text{"K "}, \text{"Heo"}), (\text{"K H"}, \text{"eo"}), \dots\}$

  $\text{At}^{-1}(\text{"K"}) = \{(x, 0)\}$

  $\text{SubStr}^{-1}(\text{" Heo"}) = \{(x, 7, 4), (x, 7, 5), \dots, (x, 7, 10)\}$

  $\text{IndexOf}^{-1}(7) = \{(x, \text{" "}, 0)\},$

  $\text{Length}^{-1}(10) = \{x\}$

  > (x, 7, n) where n > 10 never possible
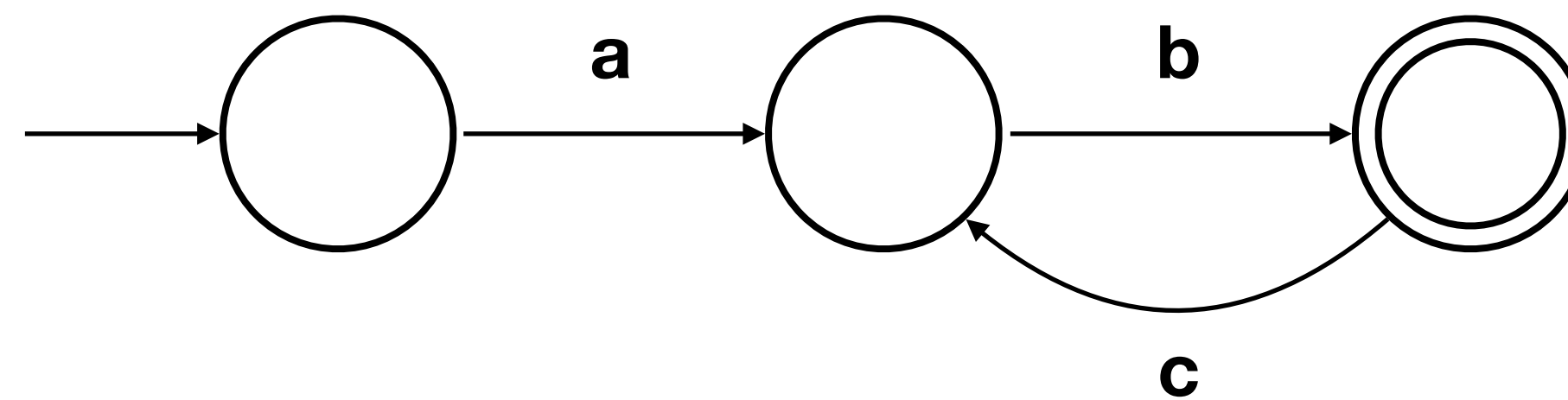 > according to the grammar

Pros and Cons

- Pros: efficient

  - Applications: Excel, VSCode, etc

  - See https://www.microsoft.com/en-us/research/group/prose/

- Cons: not always applicable

  - Efficiently computable inverse function

  - Finite inverse set

Representation-based Search

- Idea:

  - Build a data structure that concisely represents a set of programs

  - Extract solutions from that data structure

- Two well-known methods

  - Version space algebra (VSA)

  - Finite tree automata (FTA)

Automata

- Abstract models of machines

  - Computation: given an input, move through a series of states

  - Interest: the computation eventually halts at certain final states

- Many instances

  - Finite automata, push-down automata, …, Turing machine

- Example

S → aA
A → bB
B → cA
B → ϵ
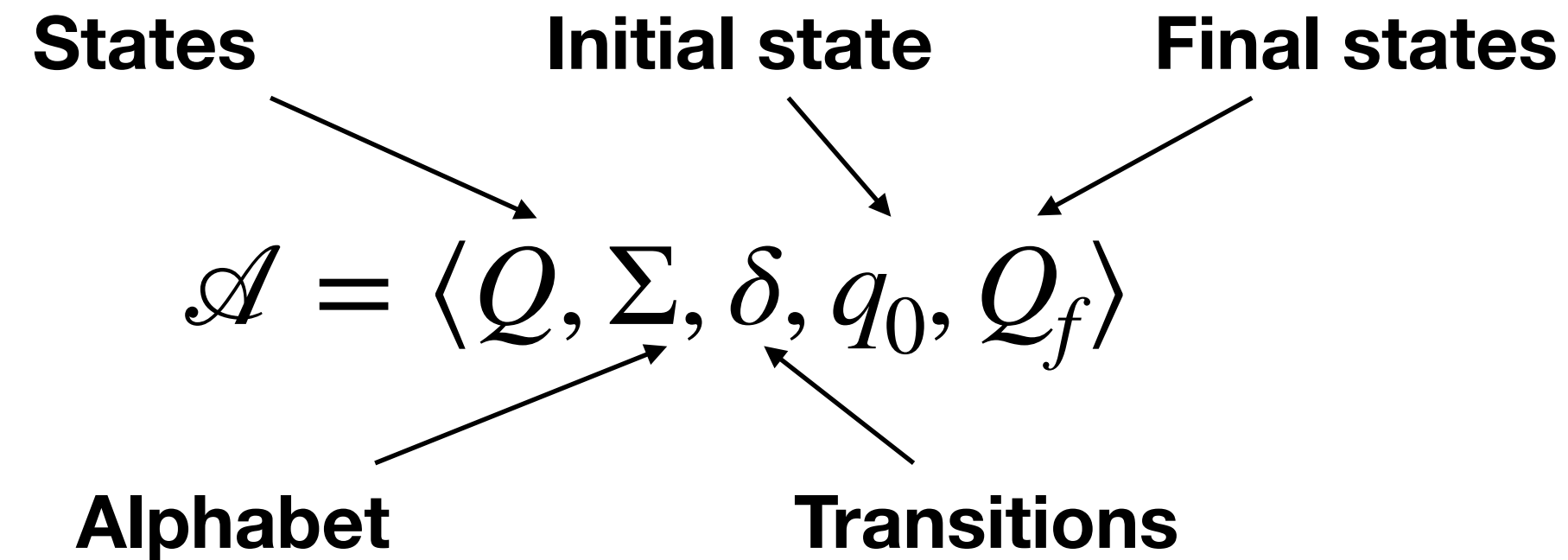
{ab, abc, abcc, abcc, …} : abc*

Example: Finite Automata

States **Initial state** **Final states**

$$\mathscr{A} = \langle Q, \Sigma, \delta, q_0, Q_f \rangle$$

Alphabet **Transitions**

- $Q = \{q_0, q_1, q_2\}$ and $Q_f = \{q_2\}$

- $\Sigma = \{a, b, c\}$

- $\delta = \{(q_0, a, q_1), (q_1, b, q_2), (q_2, c, q_1)\}$
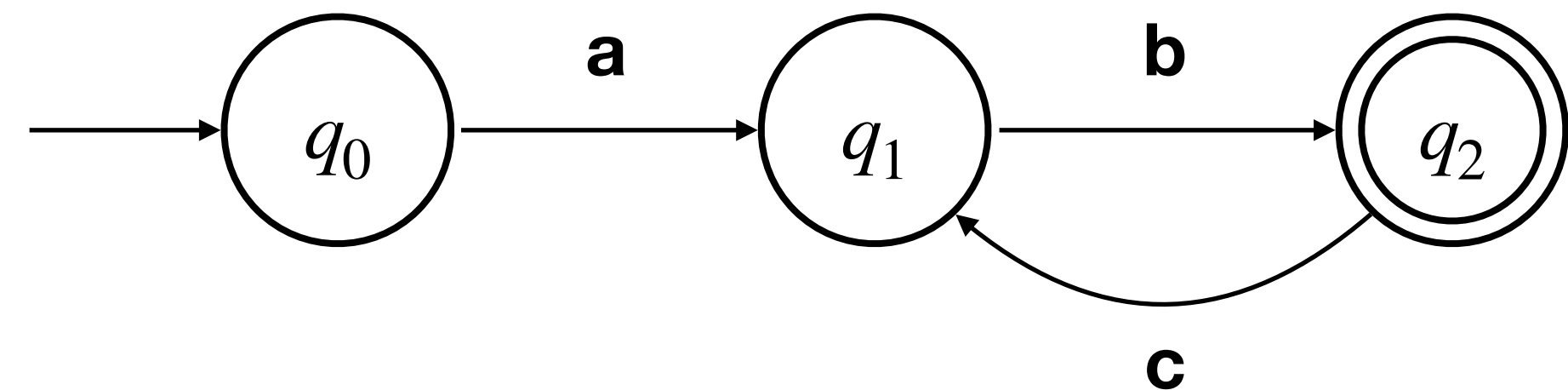
Why Automata in Synthesis?

- An automaton corresponds to a grammar

  - I.e., a set of input strings accepted by the automaton (or the grammar)

- A compact data structure for a set of programs

- Idea: bottom-up search via automata

  - Build the smallest automaton corresponding to a subset of the input grammar

  - Grow the automaton gradually according to the grammar
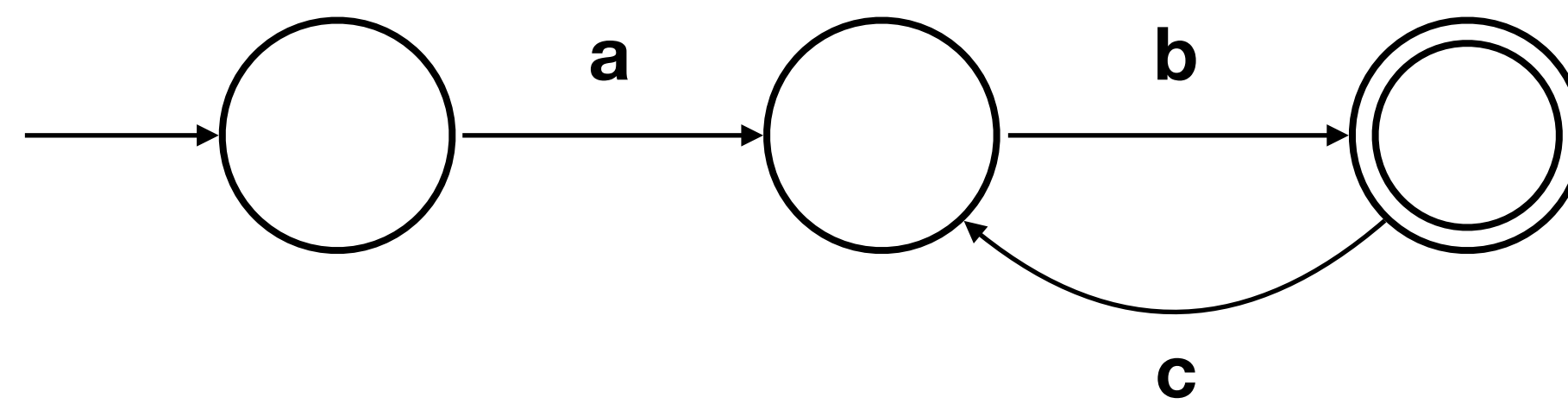
{ab, abc, abcc, abcc, …} : abc*

S → aA
A → bB
B → cA
B → ϵ

Example

Specification

Find a function $f(x)$ where $f(1) = 9$

Grammar

$N \rightarrow \texttt{id}(V) \mid N + T \mid N \times T$
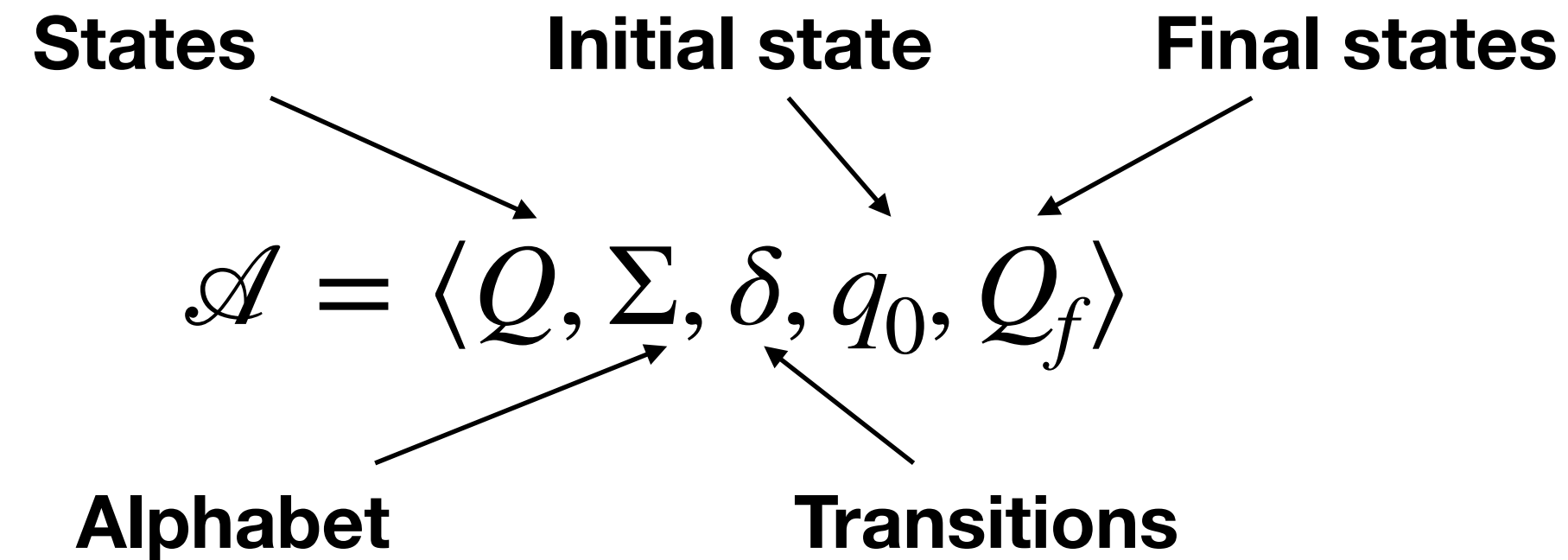
$T \rightarrow 2 \mid 3$

$V \rightarrow x$

Example

```
id(x) * 3 * 3
id(x) + 2 + 3 + 3
```

Finite Tree Automata

States **Initial state** **Final states**

$$\mathscr{A} = \langle Q, \Sigma, \delta, q_0, Q_f \rangle$$

Alphabet **Transitions**

- Example

Find a function $f(x)$ where $f(1) = 9$

$N \to \mathtt{id}(V) \mid N + T \mid N \times T$
$T \to 2 \mid 3$
$V \to x$

$Q = \{N, T, V\} \times \mathbb{N}$

$Q_f = \{\langle N, 9 \rangle\}$

$\Sigma = \{\mathtt{id}, +, \times\}$

$f(q_1, \dots, q_n) \to q$

$\delta = \{ \ \mathtt{id}(\langle V, 1 \rangle) \to \langle N, 1 \rangle$

$+(\langle N, 1 \rangle, \langle T, 2 \rangle) \to \langle N, 3 \rangle$

$\times(\langle N, 1 \rangle, \langle T, 2 \rangle) \to \langle N, 2 \rangle \quad \dots \ \}$

Bottom-up Search with FTA

Specification

　　Find a function $f(x)$ where $f(1) = 9$

Grammar

⭕ $N \to \mathtt{id}(V) \mid N + T \mid N \times T$

🟥 $T \to 2 \mid 3$

🔷 $V \to x$

$$\delta = \{\ \mathtt{id}(\langle V, 1 \rangle) \to \langle N, 1 \rangle$$
$$+(\langle N,\ 1 \rangle, \langle T,\ 2 \rangle) \to \langle N,\ 3 \rangle$$
$$\times(\langle N,\ 1 \rangle, \langle T,\ 2 \rangle) \to \langle N,\ 2 \rangle\ \ \dots\ \}$$

$$\mathtt{id}(\langle V, 1 \rangle) \to \langle N, 1 \rangle$$

Bottom-up Search with FTA

Specification

Find a function $f(x)$ where $f(1) = 9$

Grammar

○ $N \rightarrow \mathtt{id}(V) \mid N + T \mid N \times T$

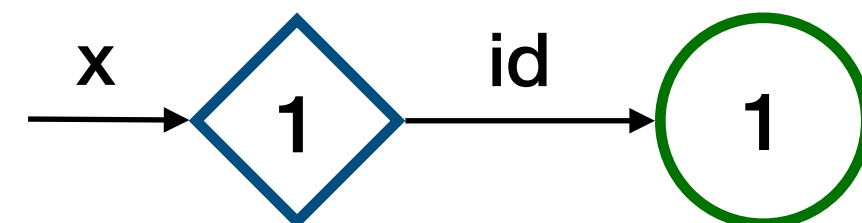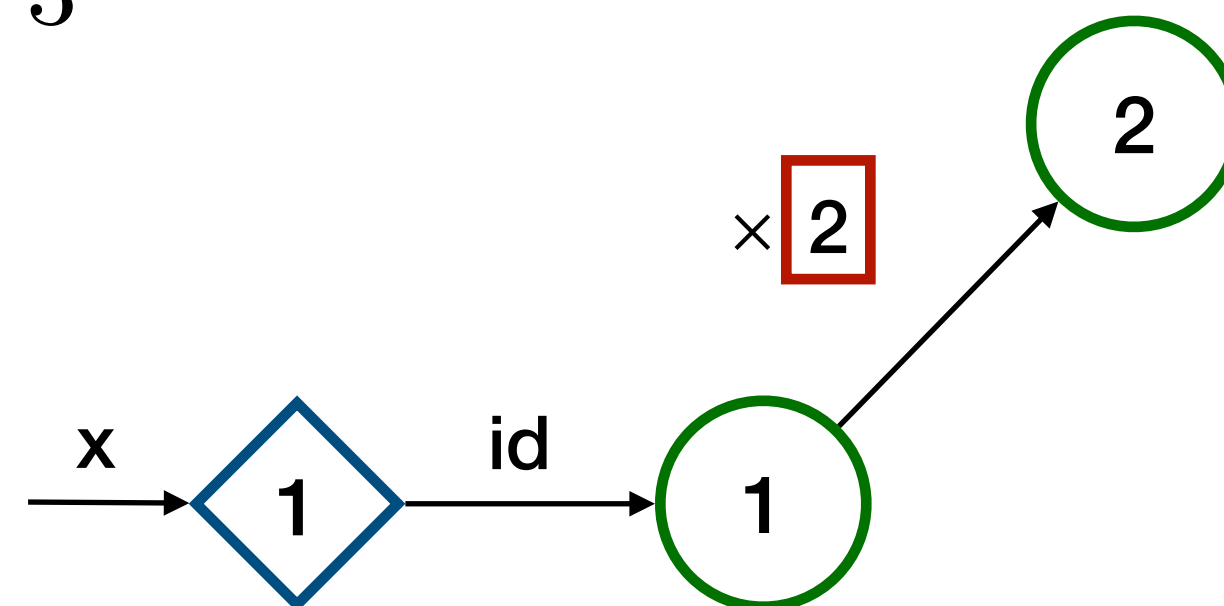□ $T \rightarrow 2 \mid 3$

◇ $V \rightarrow x$

$\delta = \{\ \mathtt{id}(\langle V, 1 \rangle) \rightarrow \langle N, 1 \rangle$

$+ (\langle N,\ 1 \rangle, \langle T,\ 2 \rangle) \rightarrow \langle N,\ 3 \rangle$

$\times (\langle N,\ 1 \rangle, \langle T,\ 2 \rangle) \rightarrow \langle N,\ 2 \rangle\ \ \dots\ \}$

$\times (\langle N,\ 1 \rangle, \langle T,\ 2 \rangle) \rightarrow \langle N,\ 2 \rangle$

Bottom-up Search with FTA

Specification

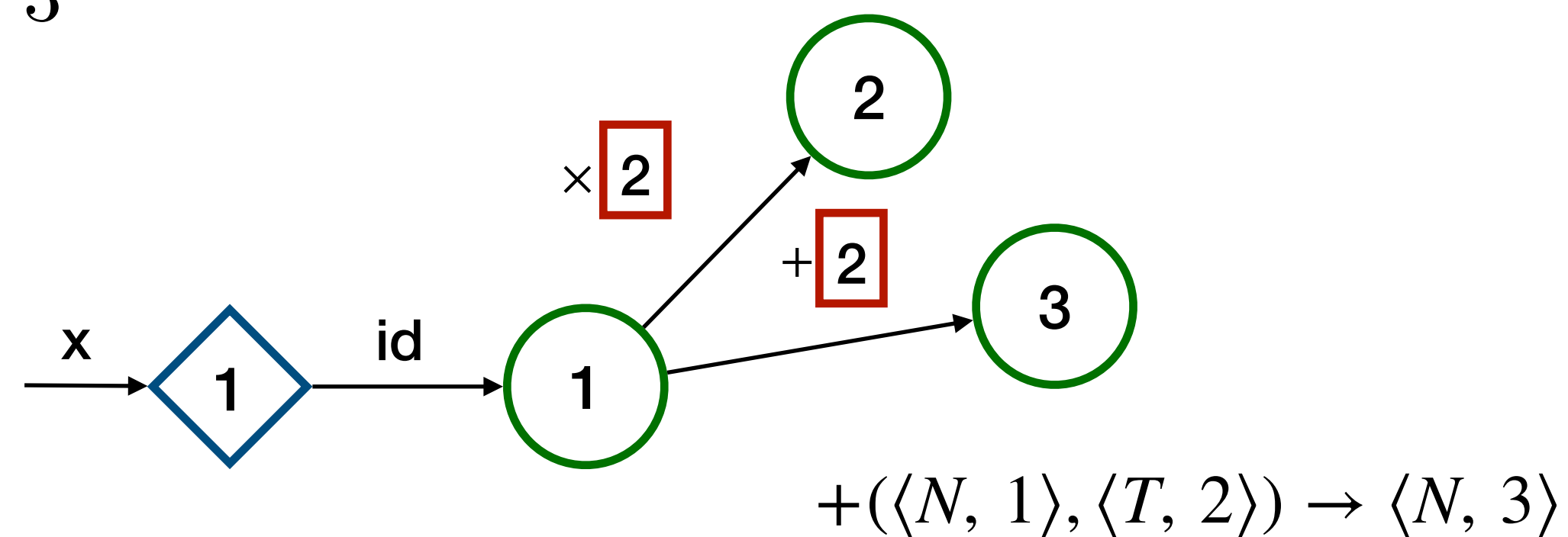Find a function $f(x)$ where $f(1) = 9$

$$\delta = \{\ \mathtt{id}(\langle V,1\rangle) \to \langle N,1\rangle$$
$$+(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \to \langle N,\ 3\rangle$$
$$\times(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \to \langle N,\ 2\rangle\ \textbf{...}\ \}$$

Grammar

$N \to \mathtt{id}(V) \mid N + T \mid N \times T$

$T \to 2 \mid 3$

$V \to x$

$$+(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \to \langle N,\ 3\rangle$$

Bottom-up Search with FTA
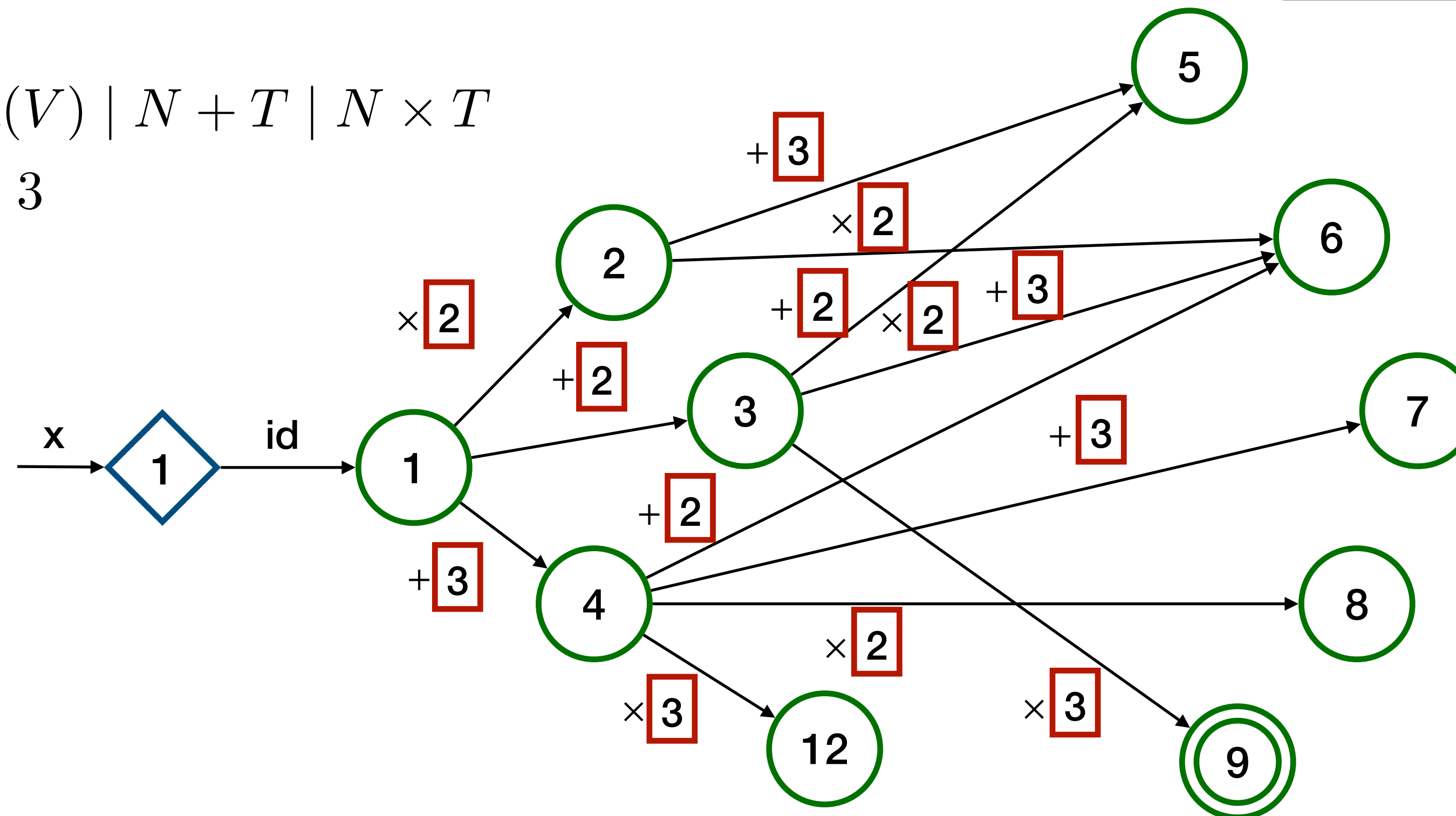
Specification

Find a function $f(x)$ where $f(1) = 9$

$$\delta = \{ \; \mathtt{id}(\langle V,1 \rangle) \to \langle N,1 \rangle$$
$$+(\langle N, 1 \rangle, \langle T, 2 \rangle) \to \langle N, 3 \rangle$$
$$\times(\langle N, 1 \rangle, \langle T, 2 \rangle) \to \langle N, 2 \rangle \; \dots \}$$

Grammar

◯ $N \to \mathtt{id}(V) \mid N + T \mid N \times T$

▢ $T \to 2 \mid 3$

◇ $V \to x$

Summary

- Representation-based search

  - Search with space-efficient data structure

  - Represent multiple programs within a simple representation

- Combination with other search strategies

  - Version space algebra + top-down search (TDP)

  - Finite tree automata + bottom-up search